几种经典搜索算法研究与应用①

欧阳圣，胡望宇

(湖南大学 软件学院，长沙 410082)

摘 要：搜索技术是人工智能的基本技术之一，在人工智能各应用领域中被广泛地使用。而搜索技术的核心是搜索算法，而所有的搜索算法的优化主要是在经典的搜索算法上改进得来。故研究经典搜索算法有非常重要的理论价值和实际应用价值。通过对几种经典搜索算法的研究，分析，总结，使得知识形成体系，便于更好的学习和研究。最后将几种算法进行比较，列出各自优缺点，便于选择合适的算法解决相关的实际问题。

关键词：广度优先搜索；深度优先搜索；回溯法；双向广度优先；分支定界；A*算法

Research and Application of Several Classical Search Algorithms

OU-YANG Sheng, HU Wang-Yu

(Software College, Hunan University, Changsha 410082, China)

Abstract: Search technology is one of the basic technology of artificial intelligence. In the various application areas of artificial intelligence, it has been widely used. The search algorithm is the core of search technology, and all of the search algorithm optimization are mainly in the classic search algorithms to improve them. Therefore, the classic study of search algorithms has a very important theoretical value and practical application value. In this paper, several classical search algorithms for research, analysis, conclusion are studied, making knowledge formation a system and facilitating better learning and research. Finally, the article compares several algorithms listing their strengths and weaknesses so that it is easy to select the appropriate algorithm to solve relevant practical problems.

Keywords: BFS; DFS; backtracking; bidirectional breadth-first; branch and bound; A* algorithm

1 引言

计算机求解的问题中，有许多问题是无法用数学公式进行计算推导采用模拟方法来找出答案的。这样的问题往往需要我们根据问题所给定的一些条件，在问题的所有可能解中用某种方式找出问题的解来，这就是所谓的搜索法或搜索技术。搜索方法作为人工智能的四大核心技术之一，不但在人工智能的各个领域中得到了广泛的应用，而且随着人工智能应用的普及，已经大量渗透到人们的日常生活中。

通常用搜索技术解决的问题可以分成两类：一类问题是给定初始结点，要求找出符合约束条件的目标结点；另一类问题是给出初始结点和目标结点，找出一条从初始结点到达目标结点的路径。

搜索技术可分为盲目搜索和启发式搜索。通常把树状盲目搜索称为穷举式搜索，它通过把需要解决问题的所有可能情况逐一试验来找出符合条件的解的方法。启发式搜索就是在状态空间中的搜索对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标。这样可以省略大量无谓的搜索路径，提高了效率。在启发式搜索中，对位置的估价是十分重要的。采用了不同的估价可以有不同的效果。

所有的搜索算法从其最终的算法实现上来看，都可以划分成两个部分——控制结构和产生系统，而所有的算法的优化和改进主要都是通过修改其控制结构来完成的。

常见的搜索算法有广度优先搜索法、深度优先搜索法、回溯法、双向广度优先搜索法、分支定界法、A*算法。下面就从两方面讨论搜索算法。

① 收稿时间:2010-08-31;收到修改稿时间:2010-10-12

2 基本的搜索算法

2.1 广度优先搜索

(1) 广度优先搜索[1,2]是一种相当常用的图论算法，其特点是：每次搜索指定点，并将其所有未访问过的邻近节点加入搜索队列，循环搜索过程直到队列为空。算法描述如下：

① 将起始节点放入队列尾部。

② While(队列不为空）

取得并删除队列首节点 Node，处理该节点 Node 把 Node 的未处理相邻节点加入队列尾部。

(2) 使用该算法注意的问题：

① 使用该算法关键的数据结构为：队列，队列保证了广度优先，并且每个节点都被处理到。

② 新加入的节点一般要是未处理过的，所以某些情况下最初要对所有节点进行标记。

③ 广度优先在实际使用时，很多情况已超出图论的范围，将新节点加入队列的条件不再局限于相邻节点这个概念。例如，使用广度优先的网络爬虫在抓取网页时，会把一个链接指向的网页中的所有 URL 加入队列供后续处理。

(3) 广度优先搜索一个典型实例：分油问题

一个一斤的瓶子装满油，另有一个七两和一个三两的空瓶，再没有其它工具。只用这三个瓶子怎样精确地把一斤油分成两个半斤油。

选择广度优先算法来求解分油问题可以得到通过最少步骤完成分油的最优解。其算法描述如下：

① 用 unVisitedBttsArr 表示初始节点列表(待扩展，此为一个动态数组) 。

② 如果 unVisitedBttsArr 为空集，则退出并给出失败信号。

③ n 取为 unVisitedBttsArr 的第一个节点，并在 unVisitedBttsArr 中删除节点 n，放入已访问节点列表 haveVisitedBttsArr。

④ 如果 n 为目标节点，则退出并给出成功信号。

⑤ 否则，将 n 的子节点加到 N 的末尾，并返回②步。

2.2 深度优先搜索

深度优先搜索[1,2]方法从初始节点开始, 顺序扩展生成下一级各子节点,放在 OPEN 表中已有的节点前面(实现后生成的子节点先扩展), 然后从 OPEN 表中提取最前的一个节点检查是否是目标节点, 否则再扩展, 再重复上述操作。这种方法每一次扩展最晚生成的子节点, 沿着最晚生成的子节点分支, 逐级纵向深入发展。这种方法搜索一旦进入某个分支, 就将沿着该分支一直向下搜索。如果目标节点恰好在此分支上 ,则可较快地得到解。但是,如果目标节点不在此分支上, 不回溯就不可能得到解。所以,深度优先搜索是不完备的,只是推理步骤。如果回溯, 不难证明其平均效率与广度优先搜索法相同。因此,深度优先搜索法如果没有启发信息,很难有实用价值。下面来看一个可以采用深度优先搜索算法求解的例子。

例如：骑士游历问题

设有一个 n*m 的棋盘，在棋盘上任一点有一个中国象棋马，马走的规则为：1.马走日字??2.马只能向右走。当 n，m 输入之后，找出一条从左下到右上角的路径。例如：输入 n=4，m=4，输出：路径的格式：（1，1）—>(2,3)->(4,4)，若不存在路径，则输出“NO”。

算法分析：我们以 4*4 的棋盘为例进行分析，用树形结构表示马走的所有过程，求从起点到终点的路径，实际上就是从根节点开始深度优先搜索这棵树。马从(1，1)开始，按深度优先搜索法，走一步到达(2，3)，判断是否到达终点，若没有，则继续向前走，在走一步到达(4，4)，然后判断是否到达终点，若到达则退出，搜索过程结束。为了减少搜索次数，在马走的过程中，判断下一步所走的位置是否在棋盘上，如果不在棋盘上，则选择另一条路径再走。

2.3 回溯法

回溯算法[3,4]也叫试探法，它是一种系统地搜索问题的解的方法。回溯算法的基本思想是：从一条路往前走，能进则进，不能进则退回来，换一条路再试。用回溯算法解决问题的一般步骤为：

① 定义一个解空间，它包含问题的解。

② 利用适于搜索的方法组织解空间。

③ 利用深度优先法搜索解空间。

④ 利用限界函数避免移动到不可能产生解的子空间。

下面介绍一个典型的回溯算法问题。

有 2n 个人排队购一件价为 0.5 元的商品，其中一半人拿一张 1 元人民币，另一半人拿一张 0.5 元的人民币，要使售货员在售货中，不发生找钱困难，问这 2n 个人应该如何排队？找出所有排队的方案。（售货员一开始没有准备零钱）分析：

(1) 根据题意可以看出，要使售货员在售货中，不发生找钱困难，则在排队中，应该是任何情况下，持0.5 元的排在前面的人数多于持 1 元的人数。

(2) 该问题可以用二进制数表示：用 0 表示持 0.5 元的人，用 1 表示持 1 元的人，那么 2n 个人排队问题化为 2n 个 0、1 的排列问题。这里我们用数组 B[1.. 2n]存放持币情况。

(3) 设 k 是 B 数组下标指针，B[K]=0 或 B[K]=1，另用数组 d[0]、d[1]记录 0 与 1 的个数，且必须满足：n >d[0]>=d[1]。

(4) 算法：回溯搜索。

① 先将 B[1]、B[2]、……B[2n]置－1，从第一个元素开始搜索，每个元素先取 0，再取 1，即 B[K]+1→B[K]，试探新的值，若符合规则，增加一个新元素；

② 若 k<2n，则 k+1→k，试探下一个元素，若 k=2n 则输出 B[1]、B[2] ……，B[2n]。

③ 如果 B[K]的值不符合要求，则 B[K]再加 1，试探新的值，若 B[K]=2，表示第 k 个元素的所有值都搜索过，均不符合条件，只能返回到上一个元素 B[K-1]，即回溯。

④ 返回到上一个元素 k：=k-1 ，并修改 D[0]、D[1]。

(5) 直到求出所有解。

3 优化的搜索算法

3.1 双向广度搜索

广度搜索虽然可以得到最优解，但是其空间消耗增长太快。但如果从正反两个方向进行广度搜索，理想情况下可以减少二分之一的搜索量，从而提高搜索速度。

双向搜索[4]比单向搜索在数量级不断增大的时候，双向搜索所体现出的优势就非常明显了。扩展次数越多，这个差距越明显。假设一个结点产生系统呈二叉树状增长，那么扩展 n 次的代价，单向是 2^n，而双向则是 $2*2^{\frac{n}{2}}=2^{\frac{n}{2}+1}$，两者相差甚远。例如魔板问题：这个题描述的非常明确：给定初始状态和目标状态和状态产生系统，要求从初状态到末状态的最短路径，符合上述双向 BFS 的特点。很明显，我们可以用双向广度搜索来解决。对于每一个状态，有两种存储方式：

① 直接用 8 个 Byte 类型存储

② 用一个 Longint 类型压缩存储

虽然②略浪费了一些时间，但是却节省了一半的空间，对于状态比较多的情况，这是很合算的，因为占用空间变大，势必造成时间的增加，在状态较多的情况下，两者是比较平衡的。双向 BFS 比较难于掌握，比较重要的原因，是因为它易错。本来检查左右两边是否重合的过程就比较复杂，而且一旦两边扩展的结点没有查找到(端的搜索“失之交臂”)往往该程序会陷入死循环。所以实际处理的时候，一定要相当细心。

3.2 分支定界

分支定界[4,5]实际上是 A*算法的一种雏形，其对于每个扩展出来的节点给出一个预期值，如果这个预期值不如当前已经搜索出来的结果好的话，则将这个节点(包括其子节点)从解答树中删去，从而达到加快搜索速度的目的。

例如：某公司于乙城市的销售点急需一批成品，该公司成品生产基地在甲城市。甲城市与乙城市之间共有 n 座城市，互相以公路连通。甲城市、乙城市以及其它各城市之间的公路连通情况及每段公路的长度由矩阵 M1 给出。每段公路均由地方政府收取不同额度的养路费等费用，具体数额由矩阵 M2 给出。

请给出在需付养路费总额不超过 1500 的情况下，该公司货车运送其产品从甲城市到乙城市的最短运送路线。

3.2.1 解题思想

首先使用 floyid 算法求出所有结点对之间的最短路长和最小费用。然后初始化一个堆栈,首先把 0 结点放入堆栈.然后每次取出栈顶的结点进行扩展,扩展的结点放入堆栈,更新路长界。如果发现超出当前的路长界或者费用的界,则进行剪枝,然后回溯。找到一个解后,进行保存,然后回溯。如此进行,直到栈空。

3.2.2 剪枝策略

如果 currentDist + mindist[cur][49] >= distBound 和 currentCost + mincost[cur][49] >= costBound 两个条件有一个满足,则进行剪枝,回溯。

3.2.3 程序实现流程说明

(1) 首先将 M1, M2 的数据读入两个 50×50 的数组。

(2) 用 floyid 算法求出所有点对之间的最短路长,和最小费用。

(3) 声明并初始化一些变量和数据结构。

(4) 建立一个堆栈,初始化该堆栈。

(5) 取出栈顶的结点,检查它的相邻结点(从上次考虑的那个结点的下一个结点开始考虑)。确定下一个当前最优路径上的结点。被扩展的结点都被加入堆栈中。在检查的过程中,如果发现超出当前的路长界或超出费用的界,则进行”剪枝” ,然后回溯。

(6) 找到一个解后,保存改解。然后回退两步。

(7) 重复上一步的过程,直到堆栈为空。当前保存的解即为最优解。

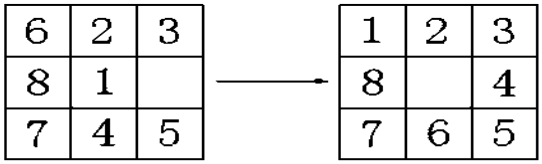
3.3 算法

A* [5,6]是一种启发式搜索，一种有序搜索，它之所以特殊完全是在它的估价函数上，如果我要求的是从初始结点到目的结点的一个最短路径（或加权代价）的可行解，那对于一个还不是目标结点的结点，我对它的评价就要从两个方面评价：第一，离目标结点有多近，越近越好；第二，离起始结点有多远，越近越好。记号[a,b]是表示结点 a 到结点 b 的实际最短路径代价。设起始结点为 S，当前结点为 n，目标结点为 G，于是 n 的实际代价应该是 f*(n)=g*(n)+h*(n)，其中 g*(n)=[S,n],h*(n)=[n,G]，对于是 g*(n)是比较容易得到的，在搜索的过程中我们可以按搜索的顺序对它进行累积计算，当然按 BFS 和 DFS 的不同，我们对它的估价 g(n)可以满足 g(n)>=g*(n)，大多可以是相等的。但是对于 h*(n)我们却了解得非常少，目标结点正是要搜索的目的，我们是不知道在哪，就更不知道从 n 到目标结点的路径代价，但是或多或少我们还是可以估计的，记估价函数 f(n)=g(n)+h(n)。我们说如果在一般的图搜索算法中应用了上面的估价函数对 OPEN 表进行排序的，就称 A 算法。在 A 算法之上，如果加上一个条件，对于所有的结点 x，都有 h(x)<=h*(x)，那就称为 A*算法。如果取 h(n)=0 同样是 A*算法，这样它就退化成了有序算法。A*算法是否成功，也就是说是否在效率上胜过蛮力搜索算法，就在于 h(n)的选取，它不能大于实际的 h*(n)，要保守一点，但越接近 h*(n)给我们的启发性就越大，是一个难把握的东西。

例如：八数码难题

问题描述为有这样一个 3*3 方阵格子：

格子上有 1-8 八个数字外加一个空格，每次只能把与空格相临的一个数字移到空格内，移动一次算作一步，给出初始状态和目标状态，求如何以最少的步数完成移动？



初始状态　　目标状态

设计 A*算法时，g(n)就取当前已移动的步数，h(n)取各个数字到目标状态中对应数字的位置的最短距离之和，这样选取的原因是，对于每一次移动，只能使一个数字改变一个相临位置，所以 h(n)步是至少需要的，所以满足 h(n)<=h*(n)。

A*的成功之处就是在选择好的 h(n)，如果实在没办法令它为 0 也是可以求得问题的解的。

4 小结

搜索算法是利用计算机的高性能来有目的的穷举一个问题的部分和所有的可能情况，从而求出问题的解的一种方法。搜索从问题性质上来看, 可分为一般搜索和博奕搜索, 从处理方法上来看, 可分为盲目搜索和启发式搜索。还可以分得更细。当所给定的问题用状态空间表示时, 则求解过程可归结为对状态空间的搜索。当问题有解时, 使用不同的搜索策略, 找到解的搜索空间范围是有区别的。一般来说, 对大空间问题, 搜索策略是要解决组合爆炸的问题。通常搜索策略的主要任务是确定如何选取规则的方式。有两种基本方式:一种是不考虑给定问题所具有的特定知识, 系统根据事先确定好的某种固定排序, 依次调用规则或随机调用规则,这实际上是盲目搜索的方法, 一般统称为无信息引导的搜索策略。另一种是考虑问题领域可应用的知识, 动态地确定规则的排序, 优先调用较合适的规则使用, 这就是通常称为启发式搜索策略或有信息引导的搜索策略。几种经典的搜索算法有各自的应用范围，现将其特点一一罗列，以便我们在学习和应用中更好的选择合适的搜索策略来解决相关的问题。

4.1 深度优先搜索和广度优先搜索

深度搜索与广度搜索的控制结构和产生系统很相似，唯一的区别在于对扩展节点选取上。由于其保留了所有的前继节点，所以在产生后继节点时可以去掉一部分重复的节点，从而提高了搜索效率。这两种算法每次都扩展一个节点的所有子节点，而不同的是，

深度搜索下一次扩展的是本次扩展出来的子节点中的一个，而广度搜索扩展的则是本次扩展的节点的兄弟节点。

广度优先搜索一般只用于找最优解，不会用于找所有解；深度优先搜索可以搜索出所有的解。

4.2 回溯法

回溯算法是所有搜索算法中最为基本的一种算法，其采用了一种“走不通就掉头”思想作为其控制结构，其相当于采用了先根遍历的方法来构造解答树，可用于找解或所有解以及最优解。

4.3 双向广度优先搜索

广度搜索虽然可以得到最优解，但是其空间消耗增长太快。但如果从正反两个方向进行广度搜索，理想情况下可以减少二分之一的搜索量，从而提高搜索速度。

4.4 分支限界

对于每个扩展出来的节点给出一个预期值，如果这个预期值不如当前已经搜索出来的结果好的话，则将这个节点(包括其子节点)从解答树中删去，从而达到加快搜索速度的目的。

4.5 算法

A*算法中更一般引入了一个估价函数 f，其定义为 f=g+h。其中 g 为到达当前节点的耗费，而 h 表示对从当前节点到达目标节点的耗费的估计。其必须满足两个条件：

1) h 必须小于等于实际的从当前节点到达目标节点的最小耗费 h*(n)。

2) f 必须保持单调递增。

A*算法的控制结构与广度搜索的十分类似，只是每次扩展的都是当前待扩展节点中 f 值最小的一个，如果扩展出来的节点与已扩展的节点重复，则删去这个节点。如果与待扩展节点重复，如果这个节点的估价函数值较小，则用其代替原待扩展节点。

参考文献

1 Kreher DL, Stinson DR. Combinationial Algorithms—Generation. Enumeration and Search. London: CRC Press, 1999: 151－186.

2 Jungnickel D. Graphs Networks and Algorithms. Translated from German by Tilla Schade Springer, 1999: 3－51.

3 吴文虎,王建德.实用算法的分析与程序设计.北京:电子工业出版社,1998.113－200.

4 严蔚敏,吴伟民.数据结构.北京:清华大学出版社,1999.

5 王晓东.算法设计与分析(C 语言版).北京:电子工业出版社,2001.162－191.

6 张德富.算法设计与分析.北京:国防工业出版社,2009.113－120,18－222,41－243.

（上接第 222 页）

5 结语

为了能有效降低后期挖掘的数据规模，尽量在不丢失重要信息的前提下降低发现访问模式的复杂度，本文提出了基于属性重要性的 WUM 数据预处理方式。该方法在引入知识信息量的基础上，给出了属性相对于属性集的重要性量化值的概念，并与页面兴趣度相结合，采用了 LRU 页面置换算法的思想，能有效地剔除噪音数据，降低 web 日志数据的规模。下一步笔者将进一步研究如何更好地设置三个阈值 minIMP、minINT 和 minTIM，并设法改进本文提出的方式，使其在时间效率上有更好的表现。

本文作者创新点：本文在引入知识的信息量的基础上，给出了单个属性相对于属性集的重要性量化值的概念，并采用了操作系统中 LRU 页面置换算法的思想，提出了基于属性重要性的 WUM 数据预处理方式。

参考文献

1 Tan PN, Steinbach M, Kumar V.数据挖掘导论.北京:人民邮电出版社,2006.

2 Mobasher B. Web usagemining and personalization. Chapman Hall & CRC Press, Baton Rouge, 2003.

3 杨明花,古志民.基于兴趣特征的 WUM 数据预处理方法.计算机应用,2006,26(10):133-134.

4 王春霞,王迤冉.WEB 日志挖掘实现网站优化.微计算机信息,2006(33):75-77.

5 陆丽娜,杨怡玲,管旭东,等.Web 日志挖掘中的数据预处理的研究.计算机工程,2000,26(4):66－67.

6 任永功,付玉,张亮.基于 web 日志的连续频繁路径挖掘算法.小型微型计算机系统,2008,29(12):2272－2276.